AMENDMENTS TO THE SPECIFICATION:

Based on the Examiner's requirement expressed in the telephone interview dated July 15, 2008, to have the present specification include more detailed description on "non-standard format" and the specific non-standard format of the exemplary embodiment described in line 12 of page 15 and line 8 of page 16 (e.g., the "register block" format further described in co-pending application 10/671,888) of the original specification, various paragraphs as follows are added. These paragraphs use the wording from co-pending application 10/671,888 (Assignee's docket YOR920030169US1), as found at:

- lines 6-8 of page 9 and lines 11-14 of page 19 (first added paragraph below);
- lines 13-15 of page 20, lines 1-2 and 13-16 of page 21, lines 13-17 of page 21, (second paragraph added below),
- lines 4-20 of page 27 (third paragraph added below).

Further, since the Examiner seemed reluctant to accept the terminology "non-standard format", definition of the standard column major format terminology is imported from lines 5-14 of page 16 of co-pending application S/N 10/671,935 (IBM docket YOR920030330US1) to become the fourth and fifth paragraphs added below.

Accordingly, the five paragraphs, previously incorporated by reference from these co-pending applications, are now expressly incorporated to accommodate the Examiner's concerns.

Therefore, please add the following five paragraphs immediately preceding the subtitle "**Level 3 Prefetching of Kernel Routines**" on page 12 of the specification:

The present invention includes using data stored in non-standard format, including, more particularly, the non-standard format described in co-pending application 10/671,888, referred to herein as "register block" format. Non-standard format, in the context of the present invention as referring to the register block format, means to store the data of one or more of the three matrices involved in Level 3 processing to be contiguous in some representation (permutation) that has optimal advantage for the L1 cache-FPU register interface of a particular architecture, rather than the standard format of row major or column major format conventionally used to store matrix data.

The present invention also is directed to Single Instruction, Multiple Data (SIMD)

machines, where $k > 1$ indicates a number of data capable of being simultaneously moved in a single instruction.

The register block data format exemplarily used in the present invention involve blocks of matrix data of size p-by-q where p and q are small integers so that the pieces of these blocks can be fitted into the registers of a particular architecture to achieve a desirable data format stored in these registers. The layout of these blocks is arbitrary. In usual cases, the p-by-q sub-blocks will be laid out either in row- or column-major format. But a key idea is that the arbitrary layout of these blocks is tailored to the architectural design of the FPU and its associated floating point registers FPUs. It should be apparent that different architectural or instruction set scenarios would require a need to lay out the blocks differently. It is intended that these different layouts, as required by unique architectural/instruction-set combinations, are considered to be in the scope of the present invention, since the register block format is intended to refer to the concept of dividing matrix data into blocks and shifting locations of these blocks to overcome "deficiencies" in computer architectural or instruction set design.

All modern programming languages (C, Fortran, etc.) store matrices as two-dimensional arrays. That is let matrix A have M rows and N columns. The standard column major format of A is as follows.

Each of the N columns of A is stored as a contiguous vector (stride 1). Each of the M rows of A is stored with consecutive elements separated by LDA (Leading Dimension of A) storage locations (Stride LDA). Let A(0,0) be stored in memory location α. The matrix element A(i,j) is stored in memory location $\alpha + i + LDA*j$. It is important to note here that stride 1 is optimal for memory accesses and that stride LDA is poor. Also, almost all level 3 linear algebra code treats rows and columns about equally.